Foundations for Extended Life Cycle Biological Models

Glen E. P. Ropella[‡], Li Yan[§] and C. Anthony Hunt[*]

Department of Biopharmaceutical Sciences, Biosystems Group,

The University of California, San Francisco, CA 94143, USA

ABSTRACT

Introduction: To expand our ability to test current concepts about system and organ function within organisms in normal and disease states we need a new class of discrete, event-driven simulation models that achieve a higher level of biological realism across multiple scales, while being sufficiently flexible to represent different *aspects* of the biology. Here we provide the first description of such models, one that is focused on the rat liver. We use a middle-out design strategy that begins with primary parenchymal units. The models are sufficiently flexible to represent different *aspects* of hepatic biology, including heterogeneous microenvironments. Model components are designed to be easily joined and disconnected, and to be replaceable and reusable. The models function within a multitier, in silico apparatus designed to support iterative experimentation on models that will have extended life cycles.

Results: Validation uses two sets of solute outflow profile data from experiments on isolated perfused rat livers (IPRL). One set, providing a measure of intrasubject variability, consists of four repeat experiments on the same perfused rat liver. The other set consists of one solute outflow profile from each of six different livers under identical experimental conditions. In both cases the solute is ^{14}C-sucrose. The in silico system contains the functional unit model (*ArtModel*), along with an accepted mathematical reference model (to help account for the in vitro data), and the experimental data model. The reference model—the two compartment, extended convection-dispersion model—represents the current state of the science in traditional mathematical modeling of in vitro IPRL solute outflow data. In establishing fine-grained control over the vascular geometry, the *ArtModel* makes four primary assumptions. 1. Physiologically accurate models are necessary to begin fully exploring the liver behavior space as reflected in solute outflow profiles. 2. Between terminal portal and hepatic veins, vascular structure can be represented by a directed graph. A multi-layered, fine-grained sinusoidal unit model is placed on each graph node. 3. The primary functional unit is the acinus or composites thereof. 4. The sucrose outflow profile is solely a function of the extracellular space and its geometry. We define and use a similarity measure to identify regions of parameter space that generate in silico results that are acceptably similar to the in vitro results, even though the model specification used here does not take advantage of interconnections between sinusoids. The results indicate that, although we do cover the behavior of the models' parameter space, a sinusoidal parameterization that targets the behavior space of the in vitro experiments with high specificity is difficult because the behavior space of the *ArtModel* is very much larger.

Contact: hunt@itsa.ucsf.edu

[‡] Also: Tempus Dictum, Inc.

[§] Graduate student, Joint UCSF/UCB Bioengineering Program

[*] To whom correspondence should be addressed.

INTRODUCTION

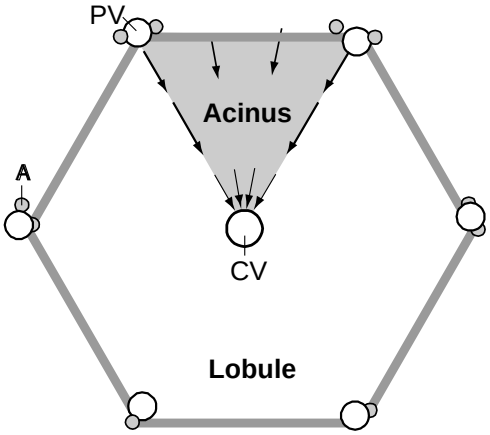
This Technical Report has six objectives.

- Present new ideas on advancing the scientific method for simulation science within computational biology.
- Define a new class of biological simulation models that can achieve a higher level of biological realism because they are event-driven.
- Use a middle-out model design strategy that begins with a primary functional unit.
- Describe models that can represent the primary parenchymal units of the rat liver and that can be extended to account for the hepatic disposition of solutes.
- Make the case that biological models need to be composed of components that are easily joined and disconnected, and that are replaceable and reusable.
- Finally, provide an example and method for building a multitier, in silico apparatus to support iterative experimentation on multiple models. We refer to it as FURM: Functional Unit Representation Method.

A goal of computational biology is to develop and apply modeling and computational simulation techniques to study biological organisms and their various organs and systems. Because biological organisms are uniquely adaptable, no one model will ever be able to represent more than a small subset of a biological system's full range of behaviors. To more realistically simulate these systems multiple models will be needed. Fortunately, recent advances in software engineering have opened a door to entirely new strategies for simulating and modeling biological systems.

Where should the modeling of organisms begin? Should we use a "bottom-up" approach and build on the knowledge of physics and chemistry, starting with molecular data and mechanisms? Should we start with phenotype and physiological systems and build down? Can we insure that the two efforts will meet and merge? Noble [Noble 2002] addresses these questions in a discussion of the pending ascendancy of computational biology. He and others [Sejnowski 2001, Bock 2002] conclude that "bottom up" and "top-down" modeling strategies will not work. We need a "middle-out" simulation strategy. But where is the middle? There is no consensus. We argue that when the aim is to simulate organisms, a logical starting point for a middle-out strategy is at the level of primary functional units. The middle-out strategy described here is designed to facilitate merger with the traditional bottom-up and top-down approaches, thus enabling linkage between molecular level events and higher level properties such as individual phenotypes.

Figure 1. A schematic of an idealized cross-section view of the hepatic acinus structure showing the direction of flow between the terminal portal vein (PV) and arterial (A) branches, and the central hepatic vein (CV).



The cell is an obvious first choice for a functional unit (e.g. leukocyte or muscle fiber). However, within organisms, groups of cells are organized in specialized primary functional units that are connected together to form various tissues and organs. Within these units cells of the same type will have different phenotypes. The examples include motor units of striated muscles (innervated by the same motor neuron), pulmonary acini, renal nephrons, and secretory acini of various glands. Here we focus on the primary functional unit of the liver. Anatomically, the liver is built of lobes that are made up of lobules. However, perfusion studies have shown that each anatomical lobule consists of several acini that are organized around terminal afferent vessels (final branches of portal vein and hepatic artery). An acinus (Fig, 1) is comprised of a network of sinusoids that separate meshworks of one cell-thick plates made up of hepatocytes, i.e. parenchymal liver cells, that perform the major metabolic functions of the liver. Intra-acinar hepatocytes exhibit location-specific properties that affect the efficacy of some of their specialized functions.

Evolution is the architect responsible for the complicated structure and function of the primary parenchymal unit of the rat liver. Ontogeny teaches that a primary unit architecture and the corresponding assembly instructions are encoded in the genome, yet we currently have little knowledge of how that is

accomplished. As a consequence, if in the near future we are given a validated in silico hepatocyte model, for example, one cannot expect any automated assembly of them to exhibit the properties observed in intact, fully functional livers. In fact, only after location is specified can cellular phenotype be specified. The functional unit model and method described here allows for the specification of location and the specialization of the cells once that location is fixed.

To test and challenge our current concepts about liver function and the role of hepatic microenvironments in normal and disease states we need a new class of models that can represent a liver as an organized assembly of primary units. These new models need to also reflect current knowledge of the heterogeneous microenvironments within primary units. The model(s) need to be sufficiently flexible to represent different *aspects* of hepatic biology at several levels of resolution. Here we describe such models. In developing these models we have adapted developments in other fields, especially those from software engineering (e.g. see [Jennings 2001]).

PRELIMINARIES

Experiments Using Computer Models

Well-designed experiments on biological components have in mind a very narrow range of plausible outcomes. Such experiments hone the specific outcome to such a fine grain, that one is intentionally examining only a very tiny slice of that component's potential behavior. The naturally occurring component is confined by the experimental design and methodology to essentially a single well-defined function or to exhibiting a single attribute. Consequently, an in silico experiment, in order to validate against the in vitro experimental results, need only model the responses that characterize that tiny, well-defined function.

However, the same biological components can be easily manipulated in completely different experimental procedures to study some other well-defined function that may bear little resemblance to the one exhibited in the other experimental context. This vague property of arbitrary interpretation is characteristic of living systems. In silico models do not have this property. In fact, many in silico models cannot easily be re-used in another experimental context because their logic is inherently tied to the experimental methodology and results that motivated their creation. The FURM represents progress in creating more realistic in silico models. It introduces methods that enable arbitrary interpretation and reuse of any given component in the system's framework.

The FURM and the in silico IPRL (isolated perfused rat liver) particular instance of that method, detailed below, is a step toward the goal of building in silico biological systems whose behavior can more closely mimic that of living biological systems at what ever level of granularity is demanded by the problem at hand. Other efforts to do this consist mostly of the integration and concurrent use of expert-designed models. The IUPS Physiome Project [Hunter 2001] and the disease models developed by Entelos [Stix 2003] are two well-known examples. Experts construct a sub-model of a specific biological component that has a pre-specified set of biological functions or roles. Several sub-models may be integrated to create one big model that is intended to behave like the corresponding biological component. If the set of biological responses simulated by the components is incomplete, then one or more *new*, redesigned sub-models must be built and integrated into the larger model. No capability for automated reparameterization or redesign is currently provided. However, FURM make progress towards such a capability.

Providing New Capabilities

In attempting to provide capabilities for automatic reparameterization or redesign FURM initially follows four steps.

1. Build a framework for running and comparing multiple models.
2. Design a mechanism for building, replicating and evolving models.
3. Use models developed for one context in other contexts.
4. Build a tournament system within which models compete and are scored based on some performance, such as their ability to help researchers discover and study biological phenomena.

The in silico IPRL is the first model implemented using this approach. The primary goal for the in silico IPRL is to demonstrate the usefulness of FURM and help us develop the method. Secondary goals are to create a validated, physiologically accurate model of liver structure and solute clearance dynamics. It makes progress on providing these four capabilities by containing a first realization of the multi-model framework in a practical context.

Refining The Concept of Simulation

Simulation can be an ambiguous word. We will use the following precise technical definition in most cases, but will stray from this usage when speaking generically about other projects or the domain of

modeling as a whole. A system is said to *simulate* another if and only if both systems can solve the same problem or answer the same questions. This definition is independent of how the systems are implemented (e.g. in biology or in computer logic). The definition is also strict in the sense that if, for a given set of specifications, system A simulates system B, then system B must be capable of simulating system A. Formally, we state:

Definition: Sets α and β are 'bisimilar" [Barwise 1996] if $\exists$ some "bisimulation relation," R, $\ni \alpha R \beta$.
Definition: A relation, R, is a "bisimulation relation" between sets α and β if:
$\forall a \subset \alpha \exists b \subset \beta \ni a R b$.
$\forall b \subset \beta \exists a \subset \alpha \ni b R a$.
$a \cap U = b \cap U$.
$U \equiv$ the set of base elements (i.e. not sets or classes, etc.).
Definition: Sets α, β, γ are "trisimilar" if $\exists$ bisimulation relations, R_1 and $R_2 \ni \alpha R_1 \beta$ and $\beta R_2 \gamma$.

Because the range of behaviors of living systems is always vastly wider than the range of behaviors of equations or engineered devices, most traditional models of biological systems will not satisfy this definition of simulation. While in general it may seem reasonable to say that a set of difference equations "simulates" some biological function, we have seen no evidence that supports the argument that the biological function "simulates" that set of difference equations. But, for FURM, it is precisely this symmetry that is desired to achieve disciplined experimental practice for in silico biological simulation. While it will not be necessary that every FURM model strictly adhere to this symmetry requirement, it is a useful heuristic for classifying models and guiding simulation-based research in computational biology.

The System and Its Environment

Naturally occurring systems never have clear or obvious boundaries. Where to draw the line between a system and its environment is always an important scientific decision, yet making the distinction can be tricky. Many modeling efforts mitigate this issue through parameterization of modules. Satisfying accepted ways of dividing up a system and coercing modules to account for specific conditions then becomes a matter of choosing the right parameter settings. Some modeling efforts include heroic attempts to make their components extensible. Once a large library of components becomes available, larger systems may be composed by integrating the parameterized components with a goal of discovering system-level phenomena. This approach to model *synthesis* will always be limited in its success because of a subtle, but critical flaw. The approach fails to concentrate on the canalizing task in simulation, which is the cycle of arbitrary synthesis, analysis, and re-synthesis that engineers and others engage in to create models that eventually satisfy their original scientific purpose. Instead, both the components and the larger system models are a priori endowed with the fundamental properties given to them by their implementation and parameterization. The components are frequently non-analytic, and in many cases they are not integrable with other components. Rather than follow this traditional approach we have chosen to open a new path. FURM (and the in silico IPRL) adopts the more open-ended compositional processes of explicit implementation in a basic programming language[1] like C or Java.

Imposing Methodological Strictures

There are a few methodological strictures in FURM that encourage the development of uniquely useful models. These strictures are enforced at the model development level and focus first on defining the experiment, the problem, and/or the questions that the models will address. Next, we put in place three particular types of model, one being the *data model*[2], the in silico representation of the experimental data, along with two dynamical system models. No restriction is put on implementing and experimenting with more models. However, two system models plus the data model are the preferred minimum[3] . They are

[1] FURM will, in the future, add to that synthetic technique by allowing the modeler to develop a genetic mechanism that enables automated structural evolution of the various systems.
[2] We use the term data model for two reasons. It is a technical term having specific meaning in the context of software engineering, and it is a reminder of one of Karl Popper's observations [Thornton 2002], that data is a prerequisite to theory and theory is a prerequisite to data. Hence, when one makes observations, those observations are biased by the devices and mind-set under which the observations were gathered. The observations simply represent a "model" of reality.
[3] The two system models may consist of anything, as long as they are representative of the referent system. Here the first system model is a pedigreed model, a traditional mathematical model that has met the test of scientific peer review. The second is a synthetic, generative, and composed model. We envision three modes: strict, moderate, and relaxed. In the first two modes FURM requires a data model. In the strict mode there are, in addition, at least two system models, one is an accepted, traditional reference model, most likely an equation-based model, and the other is formally distinct

intended to run concurrently in the same experimental framework. These methodological strictures ensure that models can interact, at least on a pure software management level (execution, starting, stopping, etc). Functionally, a model can do anything, but data taken from it must be handled by the framework, thereby ensuring that even if two distinct models carry out two unrelated functions, they rely on the same basic *ontology* defined by the experimental framework.

With this basic ontology defined implicitly by the experiments being performed, any component, regardless of what it does or how it does it, can be plugged in, and run concurrently, as long as it adheres to the restrictions put in place by the experimental plan. The result is an in silico apparatus where models can be added, removed, re-composed, dually composed, or any other arbitrary combination required by the experimental plan.

Use of Partially Ordered Sets

To achieve the desired ontological basis for in silico modeling, the formalism being used for the underlying computation must be flexible enough to tolerate (and encourage) ill-formed and informal specifications. The more sophisticated and strict formalisms for computation cannot be used because they require too much a priori knowledge about the purpose or use to which any given model might be put. It is worth noting that the goals of these more sophisticated formalisms are theoretical and targeted for formal proof or prediction as opposed to scientific exploration and discovery, or in support of decision making. Adopting a sophisticated formalism requires one to be very explicit about the assumptions being made and those assumptions must adhere to the relevant domain of the formalism. For example, Petri Nets or Systems Dynamics require any model to use the basic components that make sense in their context. By using those components and avoiding extensions, one can trust mathematical theorems that have been proven for the general case, thereby proving them for the particular model. However, complicated, more biologically realistic models become unwieldy in these formalisms because the primitives are not rich enough to concisely express complex systems. Indeed, basic programming languages have the same problem, despite their being richer in what they can express.

FURM does not require any particular formalism for the various models it manages. But the experimental framework is always formulated using Partially Ordered Sets (POSets)[4]. POSets are a generic way to specify concurrent processes with as few strictures as possible [Burns 1993], [Schneider 2001].

> Definition: A set S is called "partially ordered" if ∃ a relation, R, between S and itself ∍ the following conditions are satisfied:
>
> a R a ∀ a ∈ S (reflexivity),
>
> if a R b and b R c, then a R c, for a, b, c ∈ S (transitivity), and
>
> if a R b and b R a, then a = b (antisymmetry).

Note that not all pairs of elements in S have to be related via R.

The POSets are sets of events. An event is a message from one object to another and is considered instantaneous. Messages consist of a receiver, a method signature, and a set of method arguments. A process is a (partially ordered) set of events[5].

The concept of message passing is important to FURM because the individual models should be managed as autonomous, continuously running models that have to be integrated, managed, and measured together. This is distinct from the usual stimulus-response (input-output) concept frequently used in biological modeling. Stimulus-response posits that an object or element is inactive or in some kind of steady state or resting state as long as no other processes impinge upon it. This is clearly not the case when experimenting with most living systems. It follows (from the above definition of simulation) that it should also not be the case for in silico models of biological systems, except where it shows large gains in efficacy or efficiency to do so. In contrast, the message passing concept more accurately mimics interaction with an experimental subject (i.e. model). As a part of the FURM it also encourages the experimenter to retain the objectivity of the experiment.

Object-Oriented Design

---

from the reference model. In moderate mode the two dynamical system models can take any form. Relaxed mode can have any set of three models co-evolving.

[4] Interactions among components in living systems can be represented by directed graphs. There is a one-one correspondence between directed acyclic graphs such as Bayesian networks and POSets.

[5] See <http://www.swarm.org>.

FURM depends upon Object-Oriented Design (OOD). Objects are instances of classes with both state and behavior. However, some OOD principles are purposefully broken to support the method. For example, when encapsulation is not adhered to strictly we can support some measurement and control operations.

Agents are objects that have the ability to add, remove, and modify events. Philosophically, they are objects that have their own motivation and can *initiate* causal chains, as opposed to just participating in a sequence of events something else initiated. However, we will usually refer to agents in the prior, more generic sense. Models are agents whose purpose is to mimic some other agent or system.

Occam's Razor

Many scientists engaged in biological modeling adopt as an important guiding principle some version of the original Occam's Razor principle. Two versions are frequently encountered: *a*) use the simplest explanation consistent with the data and *b*) use the simplest model consistent with the data. An often-encountered extension of *b* is to prefer the model that gives the best "fit" to the data using the fewest number of parameters. However, a more accurate statement of the Occam's Razor principle is[6]: a problem should be stated in its basic and simplest terms. The simplest theory that fits the facts of a problem, with the fewest critical assumptions, is the one that should be selected. A simplified model should be viewed with caution if it requires that important data or properties of the system be ignored. The challenge is to systematically improve the model by accounting for as much of the available data as possible, while systematically reducing the number of assumptions.

Models often only mimic particular, isolated, phenomena. Any data set on a biological system is only a small subset of the *real* and potential behavior space. The models that treat those data have solution sets that are also very small subsets of the real behavior space. Hence, the choice of data to treat plus adherence to the Occam's razor principle (especially the two earlier versions) leads to brittle models that may serve a specific need well, but which are very limited in their power and usefulness.

If we interpret Occam's razor as being guided by the experimental procedure (problem statement) rather than the broad-spectrum phenomena or some slice of data with implications to phenomena, then the principle is more useful. Problem-driven (as opposed to data-driven) usage of Occam's razor allows for more flexible models that draw on various kinds of data, various formulations of hypotheses, various other heuristics, etc. Data set-driven usage of Occam's razor restricts the models (built to match that data) to that particular data set and its peculiarities.

METHODS

Here we go beyond the traditional methods presentation to provide sufficient rationale and background to clarify how and why each approach and method was developed.

In Silico Experimentation

Building on the preliminaries above, the following is a concise description of FURM. It is a method by which functional unit representations of extant living systems can be created, revised, evolved, and compared and contrasted with other representations of the same functional unit. This is accomplished by concurrently building, executing, and analyzing variants of articulated[7] models in the context of experimental procedures. The facility FURM provides is primarily a consequence of three fundamental guidelines.

1. Standardize interfaces to multi-paradigm, multi-mode, and cross-trophic models;
2. use discrete interactions; and
3. Enable knowledge discovery by designing for an extended life cycle (we want models that outlast a given research agenda).

These three fundamental guidelines flow down into the concrete behaviors that constitute the modeling of functional units.

Iterative Modeling

Modelers use an iterative technique to approach a valid model. Often, however, the model's evolution is ad-hoc and not reproducible. By embracing the standard of always iterating and creating new models, FURM encourages the creation of evolutionary *operators* for models. The robustness and efficacy of these operators depends, fundamentally, on a meaningful parameterization of the model. When starting

[6] See <http://www.2think.org/occams_razor.shtml> and <http://skepdic.com/occam.html>.

[7] An *articulated* model is one in which components—objects and agents—interrelate systematically, i.e., components are mutually joinable or can logically fit together in different ways.

out with a new functional unit to be modeled, an initial unrefined parameterization is chosen based on available experimental data. Components are added according to that initial parameterization. If the resulting models are not satisfactory, any given piece of the model may be modified or reparameterized with minimal impact on the other components of the model. This continues until the model provides reasonable coverage of the targeted solution space, according to the measures defined, as described below.

Once an adequate parameterization is found, the parameter space is searched for solution sets[8] that are partitioned according to defined measures. Bounds for the parameter space are then specified to indicate solution set regions for which this model validates.

When the time comes to adjust the model, there are two options, change the parameterization or change the implementation. Changes to the parameterization fundamentally alter the model and require the complete process of refinement mentioned above. Changes to the implementation, including the structure of the model, without changing the parameterization can also be made[9]. If they lead to significant shifts in the model's solution set, then those parts of the implementation that were changed should be isolated and added to the parameter set. Eventually, this process leads to a terminal set of parameter values and an operator set of combinations of those parameters, producing a prescriptive encoding and a set of evolutionary operators for the model. Different models need not have the same parameterization. They only need the same underlying space onto which their respective solution sets can be projected.

Comparing Models

The measures used for comparing and contrasting models are derived from the questions being asked. They do not derive from the structure of the model. This distinction demonstrates the difference between debugging output or instrument calibration and measurement. Measurements are made in the same way (with the same method calls or examination of state) with each of the models undergoing experimentation. That measurement should be automated, but dynamically scheduled so as to allow manual measurement. The expected result of a measurement must be sufficiently well defined to allow for prescribed assertion checking. Specifically, the system has to be able to automatically test whether the measurement generated a result that has or has not landed in an acceptable region of the solution set. In the case where a measurement cannot be made regular across different models (e.g. if the data types are incommensurate), then the encapsulated measurement device will calculate derived measures from the different models. Those derived measures are then used for model comparisons.

Validation and Verification

FURM requires a *DatModel* to represent the living, extant system. *Validation* is the process of comparing a model solution set to the *DatModel*. One or more of the above measures must compare and report on the similarity between the model solution set and the experimental data. Because FURM uses multiple models and prescribed measures that work on all models, *verification* is defined as comparing one model solution set to another. This definition is a generalization of validation.

As part of model parameterization, some estimate must be made of the change in the solution set with respect to changes in the parameters. Because the models are all run concurrently the change can be assessed qualitatively by algorithmically including a metric (or pseudo-metric) on the solution spaces. In the case where the solution space is coherent but the parameter spaces are incommensurate, the parameter spaces can be combined to create a parameter hyperspace. Sensitivity gradients would then only make sense in the context of parameter subspaces mapped to the solution space.

Aspect Oriented Modeling

All modeling begins as an intuitive, abstract, and qualitative process. Sets of model structures are considered and systematically reduced in light of available data. At some point a transition is reached and the model structures are set. The practice of *data fitting* or *curve fitting* is common in modeling that is driven by specific previously acquired data. The practice comes late, toward the end of the modeling process, and is an inherent part of refining the solution set. Typically, one is seeking the unique parameterization that enables the model to account for the largest possible fraction of the variance in a particular data set. Earlier in the model's evolution it may have changed in rather large ways. For example,

[8] *Solution set* refers to the *range* of the system, where *range* is the mathematical concept of output space and system is the transform that is effected by the algorithm, simulation, computer code, or biology.

[9] An example: In order to model flow through liver sinusoids, we currently move each solute particle according to a vector field surrounding that particle. Each particle makes a decision about which way to move by drawing a random number for movement. The same effect may be achieved by taking slices of the whole population and moving them all at once, e.g., when we expect 10% of the solute to flow backwards and the rest to flow forward.

suppose that the initial model structure is taken to be linear, yet the solution set is found to provide an unacceptably small subset of the solution set. So, the model is made non-linear. The solution set of the new, nonlinear, model is observed to cover more of the solution sets. As a consequence of this process the relationship between the parameter space and the solution space has been altered.

Aspect-Oriented Programming is used in the development of systems that are used subjectively [Czarnecki 2000]. These systems are typically broad and service multiple purposes. Consequently, any one usage is likely to exercise only a small portion of the system. Hence, AOP assists the developers in focusing on separable aspects of the system. We are extending these ideas to modeling and simulation of biological systems, and refer to it as Aspect-Oriented Modeling (AOM). AOM employs various, possibly incommensurate, models induced from different perspectives on the referent system. Rather than take the position that there is *one* ideal model for a particular perspective, AOM assumes that there are many, possibly infinite, similarly plausible models that can adequately describe the referent system and account for the available data. The modeler's task with AOM is to iteratively explore and systematically contract the space of potential models in search of a set of models that make sense in a given context, e.g., they are biologically realistic. The context for exploration is in silico experimentation. During that process the evolution and refinement of the models, up to and including any *data fitting,* are elements in that set of models, regardless of whether they validate or how similar their solution set is to those of the other models under consideration.

By adopting an AOM approach we are following the perspectival nature of the scientific method as used in biomedical research. By application of controls scientists elect to study a system from a specific perspective. It is understood that there are multiple, alternative perspectives under which the system could be viewed. For that reason the researcher may plan, as part of future experiments, to view the system from multiple perspectives. In computational biology, in silico experimental methods need to do likewise. Two specific illustrations are beneficial, one from a solute perspective, the other from the perspective of data generated by the system.

In the in vitro IPRL experiments different solutes “see” different aspects of the isolated liver under the imposed experimental perfusion conditions. Put that same solute in a different liver and that solute will encounter and report on (in the details of the out-flow data) a somewhat different environment. Change the solute or the conditions and the environment encountered can change. Resulting data will contain information on liver structure and function taken from different perspectives. The information in the data from these two solutes informs on the systems from two different perspectives, yet with several features in common. Traditionally, one would need two somewhat different models (perhaps different parameterizations of the same model) to account for the two different perspectives. Change the nature of the experimental system and the two solutes may report on still different aspects of the system. The following illustrates two different perspectives at the level of output data: 1) an outflow profile generated from a compartment model specifically designed to produce a particular curve shape versus 2) an outflow profile generated from a model designed to transition from a normal to a diseased state.

Model Granularity

All functionality is captured within FURM in a method or object. The granularity (resolution) of the functionality defines how composable[10], adaptive, and scalable the model will be. The individual models need not be fine-grained in order to run within FURM, as long as they meet the basic granularity requirements of the experimental system. But articulated models are expected to be fine-grained at all levels. There are many ways one could categorize functional granularity. The following three concepts of functionality do it well enough to sustain the FURM objectives.

The first aspect of functional granularity is fine grained execution control. Execution control refers to the responsibility for step-by-step forward movement of the aggregate POSet of events scheduled for a model or component. In the in silico IPRL simulation the POSets are indexed by “time,” although they could be indexed in other ways. The only requirement is that the index be monotonically increasing. Control is managed by that index. Each and every action taken by the constituents of any model can be subject to this control.

The second aspect of functional granularity is fine-grained spatial specificity. Any feature within an articulated model that is representing space in the referent must be subject to maximal heterogeneity. This implies that the elements in a space must be distinguishable and that the nonempty subsets of the data structures that implement a space must be accessible.

[10] We use composable to mean amenability to being composed. The idea is that the granularity of the model dictates not only the divisibility of the model, but also how well it can be joined into hierarchies (hierarchical composition) or how well it will co-exist or coordinate coherently with other models (lateral composition). The concepts of composability and articulation (footnote 7) are closely linked.

The third aspect of functional granularity is the ability to package modules arbitrarily. Understanding living systems through simulation will require perpetual revision and reengineering of models in light of new information. To facilitate this standard practice any functional relationship in FURM must be capable of being *packaged* with any other relationship. *Packaged* can mean to put them in the same class, to chain them together into a causal network, or to sequence them as they operate on some data stream. In most cases, a FURM model will provide for this *packaging* by making each event "atomic" (though not necessarily always coherent).

Encapsulating Experimental Procedures

Often, when models are developed absent sound method, the process of implementing them can lead to strange characteristics that confuse the model and may leave the experimenter confused as to how the model is to be used. Reasonable guidelines to help avoid such confusion can include developing the model so as to track the perceived characteristics of the referent system, modeling only the most salient characteristics of the referent, starting modeling by focusing on a coarse or abstract description of the referent, etc. However, the process of implementing a model can become complicated may tend to wander. depending on the attention and focus of the implementor. To bound such wandering, FURM encapsulates experimental procedures. It establishes the experimental context in which the to-be-implemented models must perform. The experimental procedure consists of the measures that will be taken on the models, the execution paradigm used, along with the reporting requirements for both the measurements and the execution of each model.

It is important to treat the experimental procedure and source materials with rigor. Even an experienced modeler can short-circuit the full experimental procedure. For example, while observing the output of a running model, the experimenter may determine that an artifact or anomaly is skewing the experimental results and make the mistake of terminating the procedure early. Or, going further, s/he may fail to keep track of key data such as source code, correlations between inputs, parameter settings, etc. In silico models are particularly sensitive to such errors because the implementations are explicitly engineered to behave in certain ways. Biological systems have no such restrictions. The fact that in silico models are predetermined may, in some circumstances, prevent them from being used to good experimental effect. That is because we have too much control over in silico models and not enough control over extant systems.

To mitigate such problems, FURM requires that the experimental procedure be codified and monitored in conjunction with the source code and data for the models. The source code is critical because it is the complete documentation of the intentions of the modeler and implementor. Binaries that generate solution sets can be tracked as well[11], in association with their source, data, and experimental procedures.

Experimental Procedures

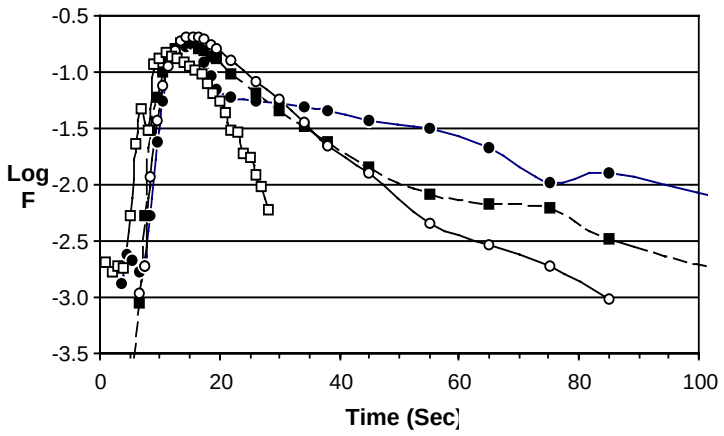
The in vitro liver perfusion protocol is detailed in Roberts and Anissimov [Roberts 1999]. Briefly, a rat liver is isolated to an apparatus and kept alive by a controlled perfusion. A solute of interest is then injected into the entering perfusate. The effluent is collected at intervals and measured to determine the amount of solute cleared by the liver. The data for time course of solute outflow contains information about the liver architecture, physiology, and subcellular biochemistry as encountered by the solute. The in silico IPRL is designed to avoid any more detail than this.

It is important to consider ratios and normalized values, rather than burdening the models with exact values and units. However, the referent isolated rat liver system, from which the data is gathered, is fundamentally dependent upon these details. Hence the measure we use to compare solution sets is of critical importance. The measure for the in silico IPRL consists of output fraction versus time.

[11] But, binaries are lossy representations of the objectives of the experiment. The source code matters more. It is also sometimes useful to track model documentation, manuals, citations, etc., along with the source, data, and solution sets; but, again, these are lossy representations of the components of the actual experiment.

Two in silico models are implemented for the in silico IPRL, the accepted, reference mathematical model (detailed below) and an articulated, functional unit model (also detailed below). Isolated parameter vectors are chosen based on some criteria for likely solutions. Input parameters and initial conditions are presented to a set of abstract dynamics, which are integrated until some stopping criteria are met. The models are run with those parameter vectors. Output is logged over time. Measurements are taken and plotted in conjunction with the experimental in vitro data. The plots from the in silico models are identified as being close, or not, to the corresponding in vitro data using some similarity measure.

Figure 2. Semi-log plot of solute outflow profiles (F = fraction of administered ^{14}C-Sucrose) for four repeat experiments carried out in sequence in the same isolated perfused rat liver. Two data sets contain data beyond the times shown.



The explicit hypothesis being tested in the in silico experiments is that the selected parameter vectors cause the model to generate output that is experimentally indistinguishable from that seen in the in vitro data[12]. The similarity measure can be used to automate the evaluation of the solution sets put forth by the models. Those results make possible automatic searches of the parameter space for regions that, together with the events elicited by the models, solve the problem (e.g., match the in vitro data).

The next step in this project will be to design and implement an evolutionary encoding for the articulated model and implement several similarity measures. This capability will allow one to arbitrarily change the shape of the solution space and evolve the models to obtain defined matches to any given data set. This capability will also allow the in silico IPRL to distinguish between experimental data and model solutions, thereby opening up the system to correlations between model structure, parameter vectors, and particular in vitro experiments.

We used two sets of experimental data. Both sets are a series of outflow profiles for ^{14}C-Sucrose, a standard indicator solute for organ perfusion studies. One set (the single-subject data), shown in Fig. 2, consists of four repeat experiments on the same perfused rat liver. The experiment was conducted as described in [Hung 2001]. These data provide a measure of intrasubject variability. The other set (the six-subject data) is taken from [Hung 2001] and consists of one outflow profile from each of six different livers under identical experimental conditions.

Three Models

The system contains three models, the articulated, functional unit model (*ArtModel*), an accepted mathematical model—the reference model (*RefModel*)—that is used to account for the in vitro data, and the in vitro experimental data model (*DatModel*). We describe each in that order.

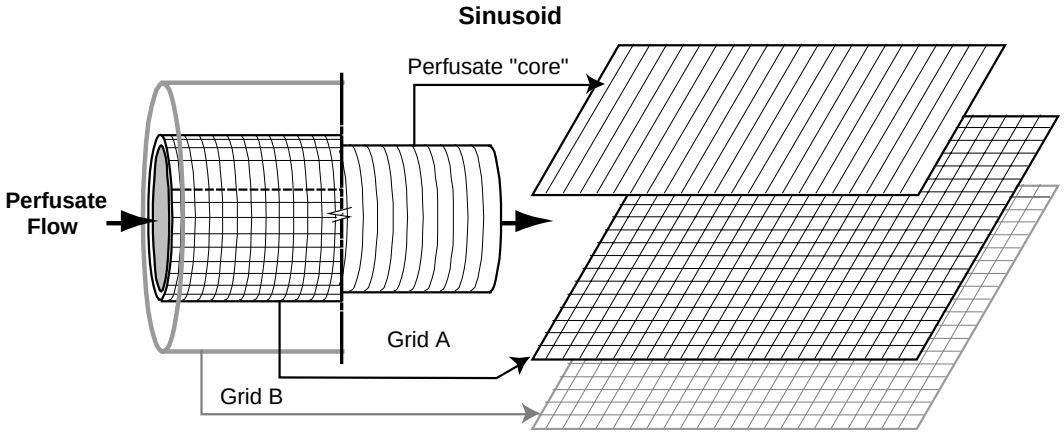
The *ArtModel* consists of a simplified, schematic representation of a "typical" hepatic acinus as depicted in Fig. 1. In rat and man it is a portion of the primary functional unit of the liver [Teutsch 1999]. Within the in silico IPRL model the functional constituents are sinusoids, perfusate, portal (PV) and central hepatic veins (CV), and solute(s). The rat acinar structure contains many tens of sinusoids, each made up of several dozen cells, hepatocytes being most numerous.

Viewed from the center of perfusate flow out, sinusoids are vascular passages modeled (basically) as a tube containing a fine-grained abstract space representing contained perfusate (the perfusate "core" in Fig. 3). Around the core is a fine-grained space (Grid A in Fig. 3) representing the inner endothelial layer of the

[12] If two sets of experimental data, each derived from fundamentally different experimental procedures, A and B, are statistically indistinguishable, we say that they exhibit *experimental indistinguishability*. Given one data set, one can not determine if it came from A or B. Here, A consists of several sets of experimental data resulting from repeat in vitro IPRL experiments. Analogously, B consists of several sets of experimental data resulting from repeat in silico IPRL experiments.

sinusoid. Another fine-grained space (Grid B) is wrapped around the inner wall. It represents the Space of Disse (SoD), hepatocytes, and the bile canaliculi.[13] It is easy to increase the number of these fine-grained spaces as well as the detail within spaces as they are needed. We can also add to the lists of objects within a space. For example, specialized Kupffer cell activity can be added to the list of Grid A objects, as can fenestrae specifications for particle sieving into the space of Disse. Active transport into and out of hepatocytes and metabolism by specific enzyme subtypes are examples of detail that can be included in Grid B or within objects within that space. Additional spaces can be added should they be needed. To account for sinusoidal heterogeneity as discussed by Anissimov et al. [Anissimov 1997], including differences in transit time and flow [Koo 1975], topographic arrangement [Miller 1979], and the different surface to volume ratios within the three acinar zones [Gumucio 1982], we defined two classes of in silico sinusoid, $\mathbf{S}_A$ and $\mathbf{S}_B$. The $\mathbf{S}_A$ sinusoids have a shorter path length and a smaller surface-to-volume ratio, and are intended to represent the *direct* sinusoids described by Koo et al. The $\mathbf{S}_B$ sinusoids have a longer path length and a larger surface-to-volume ratio, characteristic features of *branching* and *interconnected* sinusoids [Koo 1975].

Figure 3. Schematic representation of the typical in silico hepatic sinusoid.



The acinar network of sinusoids is modeled as a directed graph and this is illustrated in Fig. 4. Although five paths are illustrated in the figure, in the in silico IPRL there are many paths from the graph node representing the portal blood supply (PV) to the node representing the CV. Each path will have one to three nodes corresponding to different zones. In this study we limit the number of zones to two. Each node in Zones 1–3 has a "sinusoid"[14] assigned to it. The PV is the parent node for all nodes in Zone 1. The CV is the child node of all nodes in Zone 3. Zone 2 nodes are parents of all Zone 3 nodes, and Zone 1 nodes are parents of all Zone 2 nodes. The CV is the child node of some Zone 1 and some Zone 2 nodes. Interconnecting sinusoids may be formed, for example, by a directed edge connecting two Zone 1 or Zone 2 sinusoids (dashed connections in Fig. 4). However, for simplicity, interconnecting sinusoids are not used in the models described here.

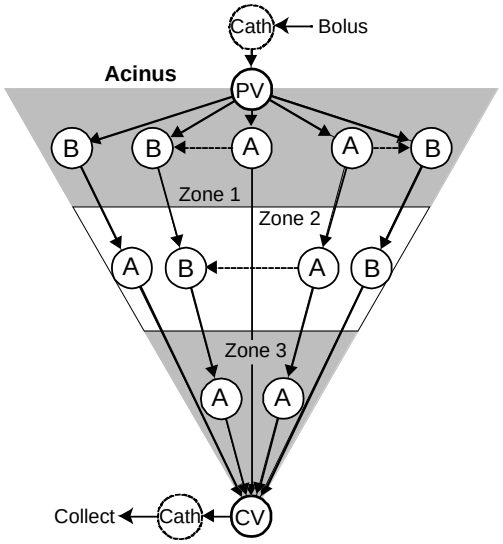
The PV and CV are referred to in the articulated model as *vasa*, and are modeled abstractly as sources and sinks for solute particles. The PV creates solute, as dictated by the experimental dosage, and distributes it to the neighboring sinusoids in Zone 1. The CV receives solute from its neighboring (parent) sinusoids.

A solute particle is a passive representation of a chemical as it moves through the in silico IPRL. At present, because the marker solute in this study is ^{14}C-sucrose, all solute is treated the same. In vitro, the solute will interact with the sinusoidal cells depending on its physicochemical and biochemical properties. In silico, solute behavior is dictated by rules specifying the relationships between solute type, grid location, and proximity to other agents, including other solute particles and hepatocytes. Those rules are specified to take into account the solute's physicochemical and biochemical properties. Once the in silico IPRL for sucrose has been adequately refined, we will shift attention to outflow data for a second solute, a third, and so on. In accounting for the data on the second solute, the sucrose-specific parameterizations will be left intact. For the in silico IPRL to adequately account for the second solute additional parameters (e.g. transporters, enzymes, new binding sites) will be called

[13] Because we are trying to build a normalized model there is no direct coupling between grid points within the fine-grained space and real measures such as sinusoid length in microns.

[14] A sinusoid is typically taken to be the full path connecting the PV to the CV. Thus the "sinusoidal" units that we place on each digraph node can represent a full sinusoid when the PV-to-CV contains only one node. However, when the path contains two or three nodes then each "sinusoidal" unit represents a portion of the sinusoid.

Figure 4. Schematic representation of an in silico acinar unit. Each circle is a node of a directed graph, typically having > 30 paths connecting the portal vein (PV) to the central hepatic vein (CV). Cath. = catheter. Sinusoid are identified with three zones as described in the text. More nodes are in Zone 1, fewer in Zone 3. Arrows indicate perfusate flow. Dashed arrows are interconnections. A: a $\boldsymbol{S}_A$ type sinusoid. B: a $\boldsymbol{S}_B$ type sinusoid (see text for details).



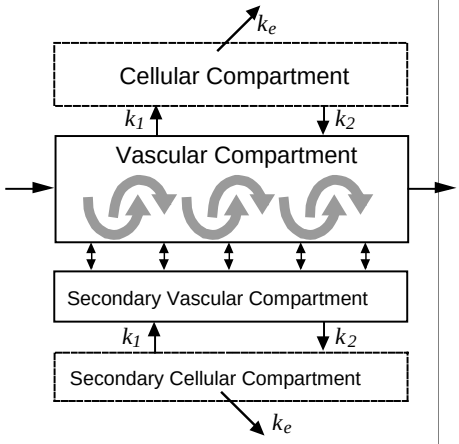
on in addition to those needed by sucrose. This is consistent with idea that sucrose and the second solute view the liver from different perspective; they "see" different aspects of the liver. Once the model for the second solute has been adequately refined, it will be possible to run experiments on simultaneous administration of both solutes. There is no obvious theoretical limit to such model *expansion.* Such model evolution makes it possible for a model and its components to have an extended life cycle.

The model runs an experiment on a single acinus, which does not scale to a model of the whole liver. To make up for this, a parameter manipulates the amount of ^{14}C-sucrose represented by a single model solute particle. When that parameter is set to 6 (or near there), the *ArtModel* represents an entire liver lobule (Fig. 1), which does scale to represent the behavior of the whole liver.

The *RefModel* represents the current state of the science in traditional mathematical modeling of in vitro IPRL solute outflow data. It is the two compartment, extended convection-dispersion model (ECD) that is diagrammed in Fig. 5 and reported by Roberts and Anissimov [Roberts 1999]. It has been used to account for the outflow profile of sucrose, others markers, and several metabolized drugs. The motivation for the model draws from signal analysis. Specifically, as depicted in Fig. 6, the solute outflow profile is thought of as a signal composed of four components: a bolus impulse, an inlet catheter effect, liver dispersion and elimination, and an outlet catheter effect. Frequency-domain representations of each of these components are derived and convolved. Next, an inverse Laplacian is calculated, giving the time-domain solution to the equations.

The focus of the ECD is the contribution to the signal (the outflow profile) made by liver dispersion and solute elimination. In the more widely used conventional convection-dispersion model (CCD), only one vascular compartment is represented. The output produced, however, fails to match the latter portions of many outflow profiles. Specifically, the tail on the experimental data often extends well above the values predicted by the CCD (that fits the data well at earlier time points). In addition, there can be a pronounced hump in the experimental data following the peak. The ECD in Fig. 6 attempts to account for these observations by hypothesizing the existence of a secondary vascular compartment. The latter is described as representing interconnected and branching sinusoids and the lag-time that such interconnectivity may introduce into the signal.

Figure 5. Schematic of extended convection-dispersion model (ECD) used by Roberts and Anissimov [Roberts 1999].

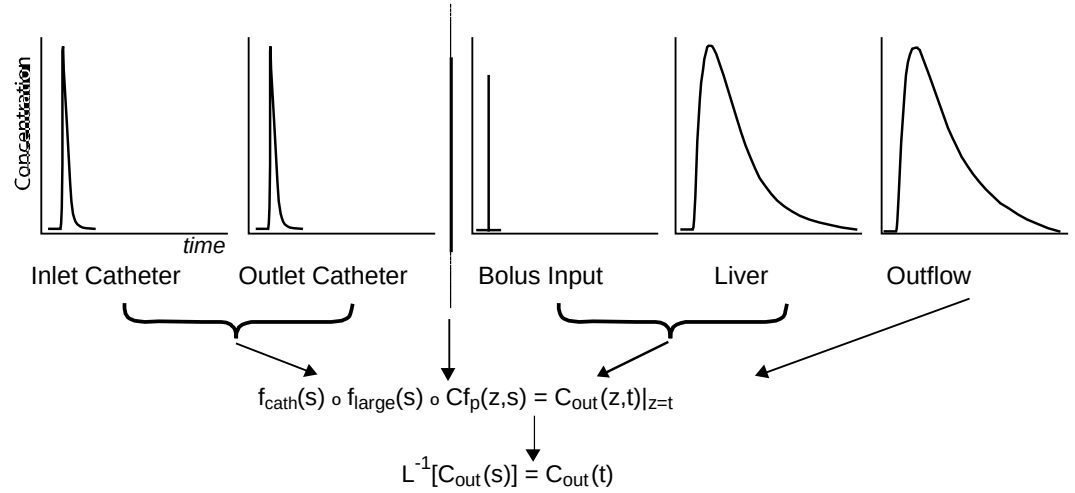


Nonlinear regression is used to fit the frequency-domain expressions in a two step process. The range of the ECD is simply a shifted and modified version of the CCD in the frequency domain. So, the first step in the nonlinear regression is to fit the CCD with equal weighting for the observations. Then the ECD is fit holding the parameters found for the CCD constant and allowing the regression to search for the modifier values. The weighting for these observations is shifted to emphasize the tail.

Because the ECD was developed to overcome subtle inadequacies in other models, the authors included a model selection criteria measure for goodness of fit, a measure of similarity between model and experimental output. This measure is based on Akaike's information criterion (AIC) [Akaike 1973], which is an estimator for the Kullback-Leibler distance [Kullback 1951]. The latter measures the difference between two models. AIC then measures the difference of a model from the referent system. The model selection criteria were then used to compare the goodness of fit for each of the models. It is critical to

hepatic elimination models to effectively predict availability and mean transit time for hepatically extracted solute. The parameters found by nonlinear regression were substituted into equations for availability and mean transit time, and the results were closely matched to the experimental data. Because the in silico IPRL is modeling hepatic efflux data for ^{14}C-sucrose, a nonextracted solute, the preceding second order test for model quality is not currently needed.

Figure 6. In the ECD the solute outflow profile is thought of as a signal composed of four components: a bolus impulse, an inlet catheter effect, liver dispersion and elimination, and an outlet catheter effect. Frequency-domain representations of each of these components are derived and convolved, as shown, and an inverse Laplacian is calculated, giving the time-domain solution to the equations.



The *DatModel* is simply a software wrapper for importing and exporting data from the in vitro IPRL. The data can be queried for points (indexed by time in this case) in between the actual data points and outside their boundary. Currently, if a time is accessed for which there is no corresponding entry in the data file, then the previous value is returned. In the case where the *DatModel* is queried before the first point, zero is returned for all values. When the *DatModel* is queried for a point after the last point, the last point is returned. Alternatively, when a smooth curve is needed, the *DatModel* can use an interpolation procedure. The in vitro data is kept in a data file formatted according to the Hierarchical Data Format version 5 (HDF5) specification[15]. An HDF5 abstraction layer queries and writes to the data file. Using this format allows the in silico IPRL to use the data file as if it were a high performance database.

The various components of FURM are detailed in the UML diagram in Figure 7.

RESULTS

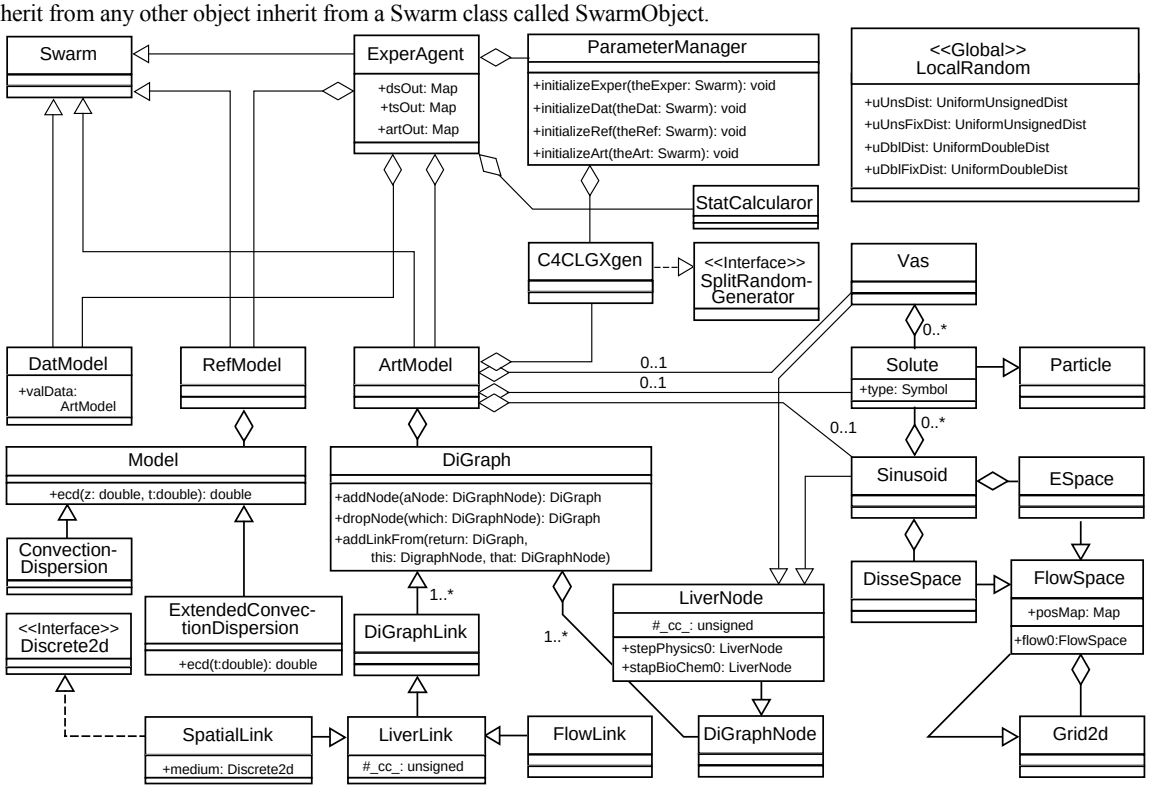
Assumptions

The primary purpose of the in silico IPRL is to provide a first example of FURM and its application. The results achieved with the in silico IPRL should be taken as preliminary examples of how various models in this new class can be developed, executed, and compared. Because their origins and types can be very different, the solution sets of the individual models are not expected to overlap completely, and may not even overlap to a large degree. It is only through the iteration of model evolution that the solution sets begin to converge. However, because the models are all developed with some intuition as to how the in vitro IPRL actually behaves, we do expect some overlap. The exceptions will lie in the particular assumptions (explicit or implicit) each model makes. The assumptions made, especially those represented by the experimental data, are often too numerous to delineate. So we focus on those salient assumptions for each model class that will shed light on the following results.

The single-subject data assumes, at the highest level, that the various rat livers are roughly equivalent, that liver viability is established by the inspection measures taken prior to and during the conduct of experiments, and that the perfusion technique and experiments do not push the liver into pathological states. The RefModel assumes that liver vascular components can be pooled, in some sense (all of them for the CCD and into two compartments for the ECD), that the physiological contributions of the various pieces of the experimental apparatus (catheter, the bolus, etc.) can be modeled as convolution in the frequency

[15] See < http://www.santafe.edu/projects/swarm/swarmdocs/refbook-java/swarm/defobj/HDF5.html> and <http://davis.lbl.gov/Manuals/HDF5/H5.intro.html>

Figure 7. An abstracted UML representation of the component structure for the ArtModel. Components that do not explicitly inherit from any other object inherit from a Swarm class called SwarmObject.



domain, and that those formulations in the frequency domain satisfy the requirements for inverting the Laplacian. The ArtModel assumes that liver function, as a whole, is an aggregate of acinus function, that sinusoids are primarily vascular objects, and that transit time for perfusate is governed by stochastic interactions between various agents inside the vascular structures in combination with the perfusion pressure at the inlet catheter and acinar PV.

These assumptions can be identified in the variation between outflow profiles generated by each model. For the DatModel, the individual sucrose profiles have very distinct shapes and features. Some have sharp peaks with fairly rapid drop-offs thereafter. Some have more blunt peaks with elevated and extended tails. Some are smoother than are others, etc. Possible sources of this variability include experimental and intersubject variation, and unidentified responses of isolated livers to the abiotic environment of the IPRL apparatus. In the solution sets for the RefModel, the assumption that the outflow profile can be prescribed by a convolution of different signals is visible in the features of the fitted curves. Those features consist primarily of a sharp rise to a peak, an inflection point just after the peak, and a relatively constant slope for the tail. The variation consists mostly of the width of the peak, when the inflection point hits, and the slope of the tail. The final assumption is that the contributors to that signal have physiological correlates.

Because the focus of this report is to present the articulated model, a more detailed treatment of the assumptions is needed. The four primary assumptions made by the ArtModel are:

1. Outflow profiles alone are lossy[16] projections of liver behavior. Models induced from such profiles are not rich enough for knowledge discovery. Physiologically accurate models are necessary to begin fully exploring the liver behavior space.
2. Hepatic vascular structure can be represented by a directed graph.
3. The primary functional unit is the acinus or composites thereof.
4. Outflow for an inert, polar solute such as sucrose is solely a function of the extracellular (vascular cavity) space and its geometry.

Of these assumptions, the latter is most susceptible to attack. Hence we have made several attempts to invalidate assumption (4) by establishing fine-grained control over the specification of the vascular geometry, and to searching for a parameterization that covers the behavior space of the in vitro IPRL experiments. Preliminary results indicate that, although the behavior space is easy to cover, finding a

[16] A computer science term: a process or procedure that results in loss of information from the original data. In the image compression field, MPEG and JPEG are examples of lossy compression techniques.

sinusoidal parameterization that targets the behavior space of the in vitro experiments with high specificity is difficult because the behavior space of the ArtModel is so much larger. We do cover that space. But, doing so precisely is difficult. Before we discuss how we arrived at this result, we need to discuss measures of similarity.

Similarity Measure

Except where pedigreed measures are not effective or known, development of similarity measures follows, at a smaller scale, the same process as model development. We typically start with a rough concept of a class of signs that indicate that any set of experiments is demonstrating essentially the same phenomena. As familiarity with the indicators grows, some of them will prove more reliable or easier to measure and the set is refined. When the set of indicators is small, we can search for ways to quantify and automate them.

Replicate in vitro IPRL experiments conducted on the same liver provide similar but not identical solute outflow profiles (Fig. 2). The same is true for the in silico experiments. In vitro, there are two main contributors to such intraindividual variability, methodological and biological. Because the living liver in these experiments must respond and adjust to its radically new ex vivo environment, it is constantly changing—making adjustments—at and below the cellular level. As a consequence, it is never the same twice. For replicate experiments (in the same liver) the coefficient of variation (CV) for fractional solute outflow during the various collection intervals ranges typically between 10 and 40%. The larger CVs are observed at earlier (pre-peak) and later times. The range of these values for each collection interval characterizes the intraindividual variability of the system for that solute. When taken together they can define a continuous band or interval bracketing the experimental data. Results of an additional experiment on that same system are expected to fall within the interval, most of the time. By definition—all things being equal—the results of replicate experiments are experimentally indistinguishable. A new set of results that falls within the interval range is defined as being experimentally indistinguishable, even if it comes from an in silico IPRL experiment. Based on the data alone, there would be no way to determine whether a data set came from an in vitro or an in silico experiment. This observation provides the basis for the first similarity measure.

The phenomena of interest in the current in silico IPRL are the various features of the outflow profile. The most important are the quick rises to a peak, the elevated and extended tails, and the transition between the two. Because the experimental data show wide variation in two of the key features, the tail and the transition between the peak and the tail, it is extremely difficult to find a quantitative description of those features from which to build an automatable similarity measure. An instantaneous, per observation, comparison is inappropriate because it is the relationship between parts of the individual curves that demonstrate the phenomenon under consideration, not necessarily the absolute magnitude at some time point. A whole-curve comparison is equally inappropriate because the three features clearly show different variances. For these reasons, we believe a multiple observation measure is warranted. Some good candidates can be derived from the Mahalanobis distance [Duckworth 1995], [De Maesschalck 2000], feature extraction methods, or using the usual statistical characteristics of the curve in a piecewise manner. We plan to explore these options for more refined similarity measures in the future.

At present, however, we are using a simple interval approach. A set of experimental outflow profiles, T, is used as training data. From this data, we calculate a distance, D, from a reference that will be the basis for a match. We then take two outflow profiles and pick one to be the reference profile, P^r. For each observation in P^r, create a lower, P^l, and an upper, P^u, bound by multiplying that observation by $(1 - D)$ and $(1 + D)$, respectively. The two curves P^l and P^u are the lower and upper bounds of a band around P^r. The two outflow profiles are deemed similar if the second profile, P, stays within the band. The distance D is the standard deviation[17] of the array of relative differences between each observation and the mean observations at that time. For the training set, T, we chose repeat experimental data on the same subject to calculate D. This allows us to arrive at an estimate of intrasubject variability.

Formally, we have $T = \{T_i\}_{i=1,M}$ where $T_i = \{[t_j, x_j]_{j=1,N}\}$

$$\bar{x}_j = \frac{\sum_i x_{i,j}}{M} \; \forall i \in [1, M], j \in [1, N]$$

$$R_{i,j} = \frac{x_{i,j} - \bar{x}_j}{\bar{x}_j}$$

D = standard deviation of $R_{i,j}$. Let P^r and P be two outflow profiles.

$$P^l = (1 - D)P^r$$

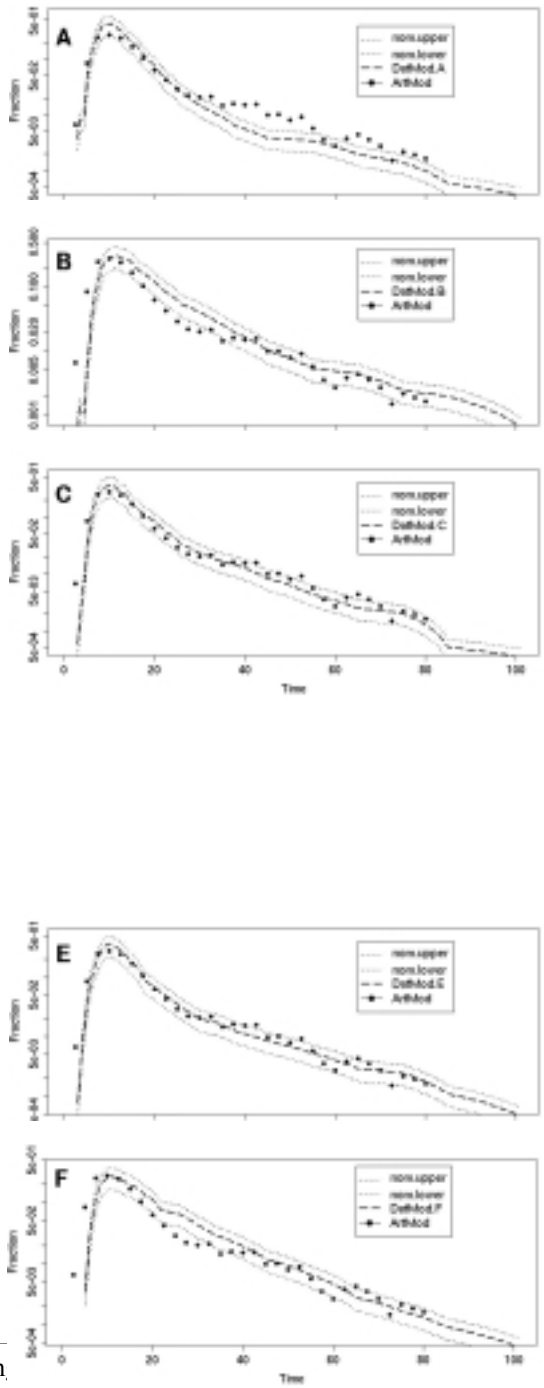
[17] For wider or narrower acceptability intervals, D may be a constant times the standard deviation.

$$P^u = (1 + D)P^r$$

Then, if $P_j < P^l{}_j$ or $P_j > P^u{}_j$, then P does not match P^r.

T = training set outflow profiles
x = output fraction
t = time (sec)
M = Number of outflow profiles
N = Number of observations per outflow profile

Figure 8. Results of a single parameterization of the in silico IPRL articulated model are shown along with each of six (A-F) individual sucrose outflow profiles along with the ± 1 standard deviation band for each (P^u and P^l).



Parameters

The in silico IPRL parameters can be divided into four distinct sets, those related to the experiment and those related to the three separate models. The articulated model has many more parameters than the other elements because it has the largest domain, range, and dynamics. The parameters for each model are tightly coupled. However, the parameters for the *ExperAgent* apply across all models. It is a design goal for the in silico IPRL to assist the modeler in prototyping models and determining what might be the parameters that are generalizable to many models. In this way, the in silico IPRL aids the modeler in classifying and analyzing classes of models. As the models evolve, parameters are expected to change semantically and migrate between being part of a model specification, an experiment specification, or being elevated to a functional unit characteristic.

To attack the hypothesis that we can achieve a good match using the above similarity measure, we parameterized the vascular geometry as described previously. The two sinusoid types, $\mathbf{S}_A$ and $\mathbf{S}_B$ have different surface area to volume specifications, different circumferences, and different lengths. The various values can be fixed or drawn at random from a specified frequency distribution. We similarly specify the ratio of $\mathbf{S}_A$ to $\mathbf{S}_B$. All that remains is specify the probability distributions under which $\mathbf{S}_A$ and $\mathbf{S}_B$ appear in the in silico acinus. Following preliminary experiments we set the $\mathbf{S}_A$: $\mathbf{S}_B$ to roughly 3. With no estimate from the literature as to which probability distribution to use, we choose the uniform distribution for sinusoid circumference and the gamma distribution for sinusoid length. We chose the latter in part because a gamma distribution is consistent with observed distributions of erythrocyte velocity in different sinusoid types in rat [Koo 1975]. Note, however, only random length values ≥ 2 are accepted. With this parameterization, it is fairly easy to find settings for the gamma distribution alpha and beta parameters to fit the quick rise to peak on each profile. The $\mathbf{S}_A$ accept solute from its parent and deliver it relatively quickly to its child (in many case the central vein) to create the rapid rise to peak. However, it is more difficult to find good parameter settin between the $\mathbf{S}_A$-dominated early portion of the curve and the $\mathbf{S}_B$-dominated final portion. With the simple two-parameter gamma distribution, the $\mathbf{S}_B$ contribute to outflow either too early or too late. So, we have

added a "shift" parameter that allows us to shift the $\mathbf{S}_B$ distribution in one or the other direction. So doing gave us enough flexibility to closely match the transition and the tail, as well.

Figure 8A-F shows a single parameterization of the in silico IPRL articulated model against each of the six individual reference data sets, as well as the ± 1 standard deviation band for each (P^u and P^l). These results were obtained using the parameter set in Listing 1, which was one the candidate solution sets found for the model specifications stated above. Even so, it is evident that this ArtModel solution set does not meet the similarity measure put forth above. A more thorough parameter sweep will reveal the closest matches of which the model is capable for each of the six data sets. That process will allow us to derive a minimal bandwidth within which ArtModel outputs must fit in each case in order to be considered a match. If we find a better solution set—a better match—then we will have validated assumption (4). If not, we will have demonstrated assumption (4) to be invalid. We will then consider what changes need to be made in order to match the in vitro experimental data more closely.

Some interesting informal observations can be made from the current system. First, the model-generated outflow profile after t = 20 sec. appears somewhat bumpy. This is because the number of runs is only 20[18]. Smoothness of the outflow profile increases with the number of runs. However, 20-30 runs are sufficient to identify profile characteristics for a given set of parameter values.

Figure A1 in the Appendix shows how these outflow profiles change as the number of experiments (model runs) is increased from 20, to 60, to 200. Figure A2 shows results from four independent repetitions of 60 experiments each using the Listing 1 parameter settings. For each repetition the random number generator seed is changed. That change alters the model specifics (e.g., placement of sinusoids on the digraph), thus providing an individualized version of the model.

Listing 1: Scheme initialization file with the parameter values for the experimental results presented in Fig. 8.

```
(list
 (cons 'parameterManager
   (make-instance 'ParameterManager
(list
 (cons 'parameterManager
    (make-instance 'ParameterManager
            #:fixedParam #f
            #:cycleLimit 100
            #:showLiveData #f
            #:showLiveAgg #t
            #:artStepsPerCycle 2
            #:artNumSinusoids 100
            #:artNumChains 50
            #:artDirSinRatio 0.8F0
            #:artTortSinRatio 0.2F0
            #:artDirSinCircMin 20
            #:artDirSinCircMax 20
            #:artDirSinLenAlpha 2.0F0
            #:artDirSinLenBeta 0.125F0
            #:artDirSinLenShift 0.0F0
            #:artTortSinCircMin 20
            #:artTortSinCircMax 20
            #:artTortSinLenAlpha 10.0F0
            #:artTortSinLenBeta 0.07F0
            #:artTortSinLenShift -40.0F0
            #:artSoluteConc 0.001F0
            #:artPerfusionRate 2500
            #:artSoluteScale 4.0D0
            #:artDosageParamA 1000
            #:artDosageParamB 1
            #:artDosageParamC 2
            #:artSinusoidTurbo 0.3D0
            #:artViewDetails #f
            #:ecdEpsilon 10^-24D0
            #:numberOfRuns 20
            #:currentRun 0
            #:runFileNameBase run
           )))
```

That change alters the model specifics(e.g., placement of sinusoids on the digraph nodes) The various experiments have shown that independent variations in the "turbo" factor (*artSinusoidTurbo*) and the geometry of the sinusoids cover much of the same solution sets. It is possible to achieve (roughly) the same outflow profile by holding geometry constant and varying the turbo factor and vice versa. Because the turbo factor is a stochastic rule operating over the movement of individual solute particles and the geometry is a stochastic rule operating over the sinusoids, there are some expected differences in discontinuities in the outflow profile tail. A low turbo factor allows the solute to diffuse and mix within the sinusoidal space. When the solute finally gets deposited in the CV, the outflow profile is smoother because the solute is not deposited with an intensity reflecting that with which it was injected. Changes in geometry affect the continuity of the tail, depending on the relationship between the concentration of solute being injected and the instantaneous carrying capacity of the input sinusoids (those in Zone 1; they are connected directly to the PV). If the carrying capacity is small compared to dose, the outflow tail will be smoother. That is because only a fraction of the solute will be taken from the portal vein at each cycle. If the carrying capacity is high compared to dose *and* the turbo factor is high (less diffusion and mixing), then the tail will contain discontinuities because solute emerges in large "clusters."

There is a second effect from the relationship of carrying capacity to dose size that is demonstrated in the model. The transit times for the solute are dictated by the size and flow dynamics of the sinusoidal spaces. One might then expect that regardless of the number of sinusoids, the progress of the solute through the network of sinusoids might be equivalently modeled by a single, large sinusoid with one aggregate space. However, the model demonstrates that flux is limited both by the dynamics of the space and by concentration gradients. If a sinusoid has a high concentration of solute at its inlet, then its carrying capacity

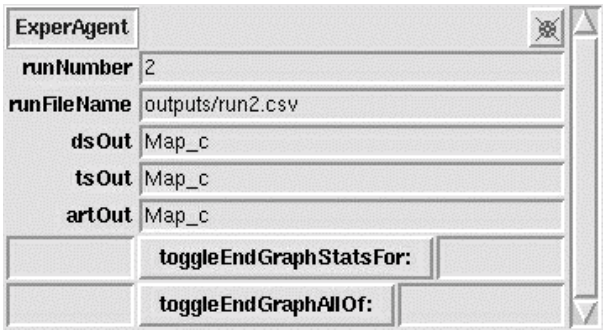
[18] An experimental result comprising 20 model runs is analogous to results one might obtain if one could conduct an in vitro perfusion experiment on only 3-to-4 liver lobules.

drops accordingly (achieved by limiting the concentration of solute at any one grid point). As the PV tries to distribute solute out to the input sinusoids, the sinusoids with the least concentration at their inlet get the higher percentage. Consequently, if there are more sinusoids (more paths in the directed graph), more solute gets distributed per cycle; but, there is no effective concentration gradient pushing the solute down the length of the sinusoid. Hence, more diffusion occurs (even though the turbo factor is constant) and more solute goes into sinusoids of various lengths. The overall effect is that flux levels out as the number of sinusoidal paths increase. In order to achieve this (realistic) behavior with a single aggregated space, one would need a highly parameterized input specification function for how the solute is distributed to the space.

ExperAgent Parameters

A screen shot of the *ExperAgent* interface is provided in Fig. 9.

Figure 9. A screen shot of the GUI widget for editing and viewing the state of the ArtModel *ExperAgent*.



numberOfRuns - Integer representing the number of times the models will be injected and the responses measured during the experiment. This dictates the number of outflow profiles generated.

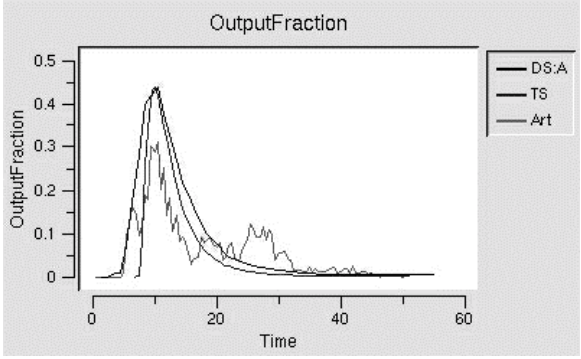
runFileNameBase - The prefix for the filenames that will contain the outputs. So, if the prefix is "run," the outputs are written to the files: run0.csv, run1.csv, ..., run[numberOfRuns].csv.

showLiveData - A boolean (0 or 1) that toggles the display of a graph during the runs showing the outflow profile for each model that is running (Fig. 10). It is possible to run the in silico IPRL so that only some models are run each time. This is useful when some of the models are invariant and some are stochastic. The current RefModel and DatModel are only plotted during the first run because they do not change from run to run.

showLiveAgg - A boolean (0 or 1) that toggles the display of a graph showing all the outflow profiles generated during an experiment (Fig. 11). For each run an outflow profile will be shown. If numberOfRuns is ten and there are two invariant models and one that varies, then, at the end of the experiment, there will be 12 outflow profile graphs.

fixedParam - A boolean (0 or 1) that toggles the stochasticity of certain components in the system. There are two pseudo-random number generator trees made available to the in silico IPRL models by the *ExperAgent*. One tree is configured so that it can be fixed by this boolean. Those components that use the fixed tree will run exactly the same for each run of the experiment if the boolean is set to 1. If the boolean is set to 0, the components that use the fixed tree will vary from run to run. For example, the *ArtModel* uses the fixed pseudo-random number generator for generating a stochastic geometry for this network, within limits. To perform an experiment where the geometry of the acinar network does not change from run to run, we set fixedParam = 1.

Figure 10. A screen shot of the *showLiveData* interface.



cycleLimit - An integer defining a stopping criterion for use by any model. If the *ArtModel* reaches the cycleLimit, the system stops. Another model can be set to control stopping, but in the in silico IPRL the ArtModel is the rate limiter.

ArtModel Parameters

artStepsPerCycle - An integer representing the number of iterations of the ArtModel that will be executed for every iteration of the ExperAgent.

artNumSinusoids - An integer representing the number of sinusoids in the acinus being modeled.

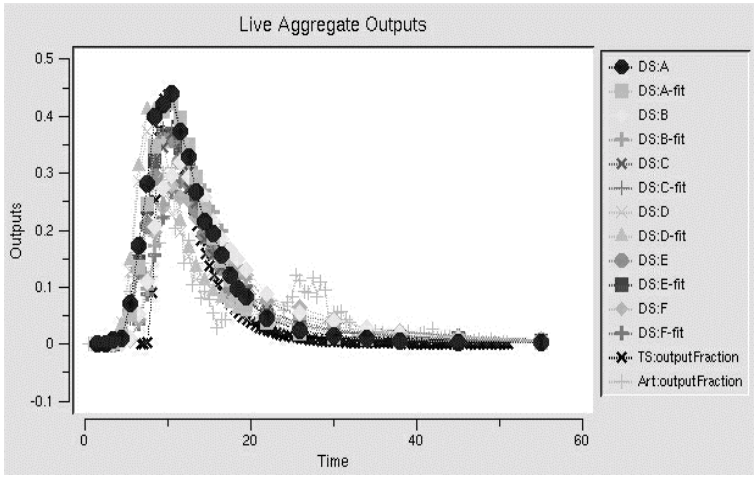
artNumChains - An integer representing the number of paths (prior to addition of interconnections), consisting of connected sinusoids, from the portal vein to the hepatic vein.

artDirSinRatio - A floating-point number indicating the percentage of artNumSinusoids that are type $\mathbf{S}_A$.

artTortSinRatio - A floating-point number indicating the percentage of artNumSinusoids that are type $\mathbf{S}_B$.

artDirSinCircMin - An integer representing the minimum circumference of an $\mathbf{S}_A$.
artDirSinCircMax - An integer representing the maximum circumference of an $\mathbf{S}_A$.
artDirSinLenAlpha - A floating-point parameter for $\mathbf{S}_A$ length distribution.
artDirSinLenBeta - A floating-point number for $\mathbf{S}_A$ length distribution.
artDirSinLenShift - A floating-point number for $\mathbf{S}_A$ length distribution.
artTortSinCircMin - An integer representing the minimum circumference of an $\mathbf{S}_B$ sinusoid.
artTortSinCircMax - An integer representing the maximum circumference of an $\mathbf{S}_B$.
artTortSinLenAlpha - A floating-point parameter for $\mathbf{S}_B$ length distribution.
artTortSinLenBeta - A floating-point number for $\mathbf{S}_B$ length distribution.
artTortSinLenShift - A floating-point number for $\mathbf{S}_B$ length distribution.
artSoluteFraction - A floating-point number indicating the percentage of perfusate that can be occupied by solute. Essentially, this parameter sets the initial concentration of solute any *Vas* can handle. If the Vas is an input, then this number dictates the initial value for the amount of solute produced. If the *Vas* is an output, then this number dictates how much solute can be cleared at each time cycle. The *Vas* are initialized with numbers derived from *artSoluteFraction*. In the bolus experiments, the input *Vas* solute flux is modified at each cycle. The output *Vas* retains the same outflow capabilities through the whole run.

Figure 11. A screen shot of the *showLiveAgg* interface showing Swarm-generated graph of output fraction for all models, including each profile in the *DatModel*. This graph is visible for the entire experiment and accumulates data for the various runs for live, side-by-side comparison. The graph is only visible when the *showLiveAgg* parameter is set to true.



artPerfusionRate - An integer representing the acinar "carrying capacity" for each cycle. This is, in effect, the maximum amount of perfusate that can enter or exit the acinus for any given cycle. It determines the absolute number of empty slots available for solute particles to occupy.

artSoluteScale - A floating point value specifying the relationship between a solute particle and a real experimental concentration of solute molecules. If this is set to 6, then each Solute object represents 6 solute molecules.

ArtDosageParamA; artDosageParamB; artDosageParamC - Integers parameterizing the dosage input function: $d(t) = A(B^C t^{C-1} e^{-Bt})/(C-1)!$, where t = current cycle.

artSinusoidTurbo - A floating point value in [0.0, 1.0] representing the tendency of solute (and perfusate) to flow from the input of any vascular structure to the output. A lower number allows more turbulence and backflow to occur. A higher number causes solute to march more directly from input to output. This number is used to initialize a vector field for the space inside the sinusoid. The vector field is implemented by placing a normalized 9-element (Moore neighborhood) probability vector at every grid point. For each solute present in the sinusoid, a pseudo-random number is drawn and the value of the draw indicates which direction the solute will move. For transitions from the core of a sinusoid to the inner wall (to Grid A in Fig. 4) and vice versa, an additional pseudo-random number is drawn to establish whether or not the solute will jump from one to the other. The probability of a jump occurring is calculated from the probability vector at the grid point under consideration and equals the probability any given solute will move sideways or backward.

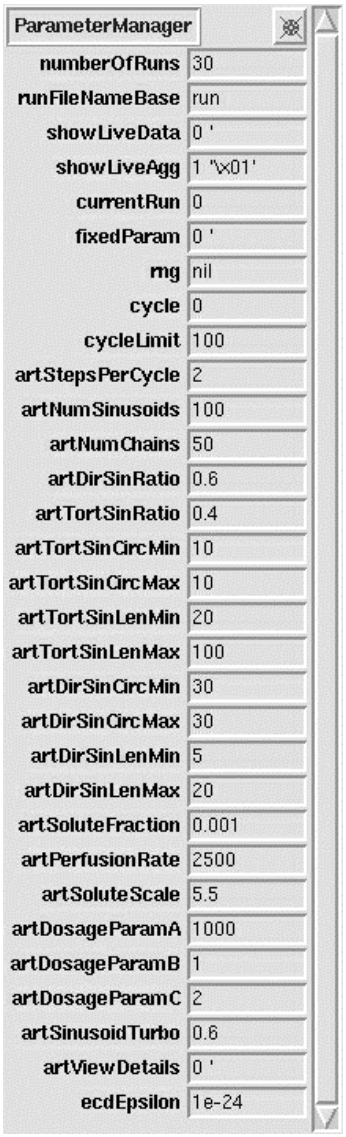
artViewDetails - A boolean (0 or 1) that toggles whether or not extra ArtModel internal reports are printed to the console window as the ArtModel is run.

RefModel and *DatModel* Parameters

The *RefModel* currently only allows users to change one parameter. The CCD and ECD models themselves have several parameters (see Fig. 4), including dosage, flow between the compartments, solute

uptake rate, transit time for solute, perfusate flow rate, etc. In the future, these parameters will be standardized and exposed in the *ParameterManager*.

Figure 12. A screen shot of the *ParameterManager* GUI widget for setting the experimental parameters. When the simulation begins, the data in these fields reflects the entries in the Scheme initialization file.



ecdEpsilon - A small, positive, real number specifying how close to zero the RefModel outputs must be to be considered equal to zero. This acts as a stopping criterion because the RefModel essentially implements an asymptotic function that can come arbitrarily close to zero without reaching it.

The current *DatModel* has no parameters. The choices about how the *DatModel* is initialized and evolves are made by the data and how they are represented. In the future, there will be some parameters that will allow changes in the way the data is used. For example, at present, if the data contains observations at time t_i and t_{i+1} and the *ExperAgent* queries for an observation between these two, then the *DatModel* returns the t_i observation. Two parameters that will be added in the future will provide for interpolation and extrapolation of the data between and beyond the observations actually contained in the data.

ParameterManager GUI and File

The parameters above can be set in two ways, with the Scheme file (Listing A) or with the GUI widget that represents the *ParameterManager* (Fig. 12). The in silico IPRL automatically initializes the ParameterManager with the contents of the Scheme file at startup. This provides the user with the ability to specify various experiments in a permanent and executable form. After the ParameterManager is initialized, the user can modify the parameters by explicitly typing in new values into the *ParameterManager* GUI widget.[19]

ProcCtrl and *ExperAgent* GUI Widgets

When the program is run, two other widgets appear: the *ProcCtrl* (stands for process controller) and the *ExperAgent ProbeMap*. The *ProcCtrl* presents the user with the START, STOP, NEXT, SAVE, and QUIT buttons. START sets the system running such that it will not stop until the end of the experiment. STOP will stop a running experiment. The execution is designed so that the experiment cannot be stopped in the middle of a run. The current run can be seen in both the *ParameterManager* widget and the *ExperAgent ProbeMap*. NEXT causes in silico IPRL to execute the next run that is queued up in the system and stop before executing any more. SAVE simply saves the positions of the windows currently on the screen so that when the system is reinvoked, those windows position themselves in the same places. QUIT stops the in silico IPRL and closes all the windows. However, the QUIT command is queued up and will not always stop the system right away It waits until the current tasks on the queue are complete before it quits.

The *ExperAgent ProbeMap* helps the user monitor the progress of the system and engage in some data visualization at the end of the experiment.

End of Experiment GUI Widgets and Output Files

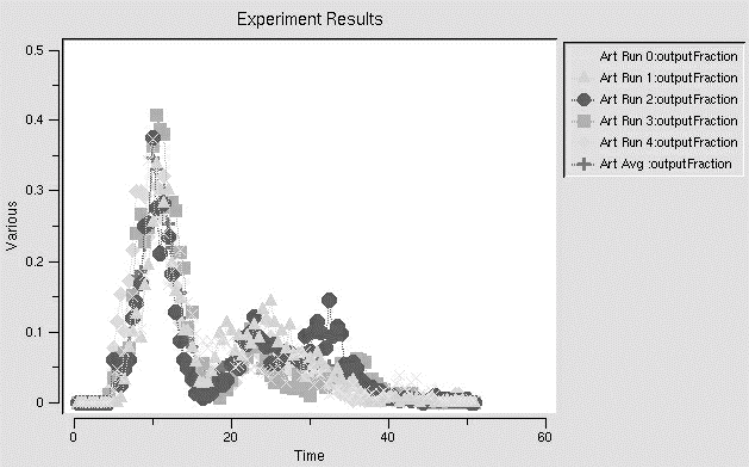
After the experiment is complete, a graph widget appears showing the outflow profiles of all the runs of the stochastic models. This is one default example of some primitive data viewing tools within the in silico IPRL[20]. The two buttons in the *ExperAgent ProbeMap*, *toggleEndGraphStatsFor*: and

[19] One must be sure to hit the <RETURN> or <ENTER> key after any changes so the program will register the change.
[20] To graph the derived measures from the *DatModel*, for example, place the pointer over the Map_c entry next to dsOut in the *ExperAgent ProbeMap*. Click Map_c and drag it so that the pointer sits inside the entry box next to the toggleEndGraphStatsFor: button, release the mouse button, and click the toggleEndGraphStatsFor: button. (If the

toggleEndGraphAllOf: allow the user to summon widgets showing the derived "statistics" calculated from each model and the entire suite of outflow profiles generated by each model on a per-model basis (Fig. 13).

Several output files result from each experiment. For each run, a file is created containing a table with columns of time, ref, art, dat1, dat2, ..., where "ref" means the output of the reference model, art is the output of the articulated model, and the various "dat" values are one or more outputs from the data model. A special output file is created for the articulated model called *art_results.csv*. This file contains a table of two columns consisting of time and art. However, the observation in this file is an average value of each of the articulated model outputs over the whole experiment, thereby representing the nominal outflow profile for the articulated model.

Figure 13. A screen shot of Swarm-generated graph of output fraction. It appears when the experiment is complete and shows outflow profiles for each run in the experiment. By default, it only shows the output from the *ArtModel* and the average of the *ArtModel* oberservations. Similar graphs of the *DatModel* and *RefModel* can be generated using the *ExperAgentProbeMap* widget.



DISCUSSION

Building functional unit models from experimental results and hypothesized descriptions of biological functions is an integral part of the way biology is studied. The models that result are useful in explanation, prediction, and hypothesis testing. They are built by a process that starts out very intuitive and qualitative and ends in very precise quantitative refinement through methods like sensitivity analysis and data fitting. The downstream methods for the modeling process are rich with disciplined and quantitative techniques. But, the upstream steps in this process are still extremely domain-specific and intuitive. FURM is intended to assist the computational biologist in bringing more discipline and more of the accumulating, research-derived information to the upstream development of widely different models and to assist in exploring the range of behaviors of any given model, as well as the referent systems, themselves.

Because models are representations or stories about what might be happening in the living system, they are composed in much the same way an artist or novelist might assemble a story. These representations do not, ultimately, have to be based directly on biological fact in order to be useful. They simply must help scientists understand and predict the behavior and function of the system being studied. In other words, they must "validate" against the tasks the scientist is trying to perform.

Discovery vs. Instantiation in Science

Scientific models are often successful in spite of large amounts of uncertainty and of ignorance of the behavior and function of the system being studied. Although the scientific method is founded solely on the concept of falsification, in practice, all yet-to-be-falsified theories can be "taken seriously" as candidate representations of reality. When a breakthrough is made and the population of yet-to-be-falsified theories drops drastically and the effectivity of the remaining theories rises, the scientific and lay communities frequently treat the event as a "discovery." The implication is that researchers have found something of apparent value by searching for it.

However, if one examines what has happened in these cases, one can think of it in an entirely different way. It may be that over time a set of arbitrary (and some random) ideas are instantiated and connected together to form a theory. There are many of these arbitrary stories documented in the literature and those that appeal to scientists (and/or to funding agencies) are studied, evolved, and tested. Some of those stories become accepted as reflecting an aspect of biological reality. What matters is whether or not,

"Experiment Results" widget is already on the screen, clicking the button once will remove it and clicking it again will display the new graph.)

by adopting or using a story, a scientist can more successfully predict and explain the phenomena perceived in the organism under study.

Scientific modelers often restrict themselves in the methods they use and in the data on which they prefer to focus. Their models often assume simplifying properties (like linearity) to make them easier to use, understand or teach. At times, sheer psychological inertia prevents new or counter-intuitive models from gaining ground or being perceived as credible. When methods are firmly based on validation (or invalidation), their origin does not matter. What matters is whether or not they are useful.

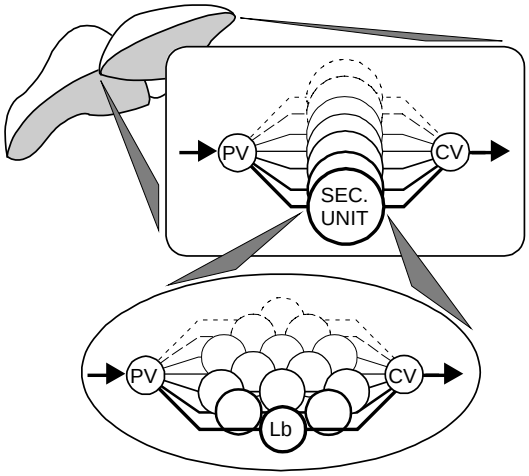
FURM, in its small way, is intended to help expand and formalize the methods by which new and strikingly variant models can be created, evaluated, and evolved. The in silico IPRL consists of an entirely new model of the liver. In and of itself, it may or may not prove useful in explanation or prediction of actual liver behavior. But, when used and evolved side-by-side with data against which to validate and trusted models with which to compare, it presents us with a vehicle for continual clarification of what may be going on within the liver without restricting the other models we may want to compose and investigate.

Higher and Lower Levels of Resolution

There is pressing need to connect genetic and molecular level details sequentially through the various levels of biological organization to whole liver and beyond to the individual and phenotype. This report describes a model of a hepatic acinus. How can FURM enable and facilitate connectivity across scales? The simulation of solute outflow data described above did not require implementation of model features at different levels of resolution. However, providing that capability (when needed) has been part of the FURM design from the start and is now briefly described.

A lobule can be represented in at least two different ways. Either implement a small set (e.g., four or more) of independent acini of the type specified in Listing 1, or increase the average number of paths in the acinar directed graph by a factor of four or more. In rat and human a network of a dozen or so lobules are known to comprise a secondary unit [Teutsch 1999]. A liver lobe is believed to be comprised of a large number of these secondary units. To represent the whole liver we connect in parallel four-to-five different size lobes. To represent a liver lobe we implement a directed graph (Fig. 14) that has multiple parallel paths connecting nodes representing the portal and hepatic veins. Each path also represents arterial blood flow. Each path contains at least one node. At each of these nodes we place a agent representing a secondary unit. Each secondary unit is similarly represented by a directed graph with lobule objects placed at each node of the multiple paths. This hierarchical detail can be present in the model, yet called on only as and when needed.

Figure 14. A schematic illustrating the hierarchical structure of the in silico IPRL model as described in the text. A liver lobe is comprised of a network of parallel secondary units (SEC. UNIT) [Teutsch 1999]; they, in turn are comprised of an organized network of lobules (Lb)—primary units, and the lobules are comprised of several of the acinar units in Fig. 4. PV: Portal vein. CV: Central hepatic vein.



The endothelial space (Fig. 3, Grid A) can include as much detail as needed to accomplish the experimental objective. A subset of grid points represents one or more Kupffer cells. Another subset represents intracellular gaps and the fenestrae of endothelial cells. Solutes or other objects that move from the perfusate core to a grid point in this space are subject to one or more lists of rules that are called into play at the next cycle step. These rule lists (in some cases decision trees) can be structured to recognize and take into account solute-specific attributes such as size and physicochemical properties. A solute object that arrives at a grid point specified to be part of an endothelial cell may (or may not), in the next time cycle, "partition into the endothelial cell." If so, the object—still at the same grid point—becomes subject to additional rules, one of which may be to "partition out of the cell" into the space of Disse or back to the sinusoidal core.

The hepatocyte space too can include as much detail as needed by the problem (Fig. 3, Grid B). Individual grid points or sets of grid points represent individual hepatocytes. A different subset of grid points represents bile canaliculi. The hepatocyte-specific actions of macromolecules, such as transporters and enzymes, and processes, such as induction of gene expression, are represented as objects or incorporated into the rules list used by specific grid points. As the list of hepatocyte functions represented within the model grows, and as the number of mobile solutes increases, then it will become simpler to add additional

grid spaces (Grids C, D…) to represent intrahepatocyte systems and functions. It is important to recognize that if a set of model capabilities is not needed for a particular in silico experimental objective, then they will not be "run." For example, if the experiments are focused on cationic solutes, then computations (objects, rules, lists, etc.) intended for anionic solutes or particles will not be fully activated. Even though the capability is in the model, they are only be used when needed.

It is likely, particularly in the early life of this system, that in order for FURM to solve a new problem, it will be necessary to add one or more new capabilities to those already present. A completely new model will nor be needed. By successively building on and revising the existing system, we extend the life cycle of the earlier systems. The more research that is done with the system, the more it can evolve. Of course system growth and expansion, which is essential, carries with it its own set of issues and problems. However, as has been demonstrated by software engineers, these are manageable issues.

In Vitro vs. In Silico

The in silico IPRL strives to replicate the experimental procedure that has provided the experimental data sets it uses. But, there are overwhelming differences between any in silico model and an in vitro experimental biological system that no amount of discipline can reconcile. Examples of these in the in silico IPRL are the computational limitations that disallow (or dramatically increase the cost of) the simulation of moles of solute and a realistic population size of primary units in the liver. However, because the in silico IPRL is validation-centric, what matters is whether or not the computation mimics the behavior of the real system, to the extent captured in the experimental data, and within the other models being used. The in silico IPRL achieves this by focusing on the design, execution, and analysis of experiments.

Experimenting on Computer Models

There seems to be a widespread attitude among some traditional biomedical scientists that computer models are not appropriate candidates as targets for experimentation. At the heart of this problem is a logical error called "inscription error" [Smith 1996], wherein it is believed that a computer model will only demonstrate behavior that its human author designed in. While this is, of course, true, it presumes that the limits of computation are short enough to prevent a computational process from ever behaving enough like an actual biological system to enable discovery of new knowledge. The results presented to us in the domains of metamathematics (Turing, Goedel, von Neuman), modern physics (relativity and quantum mechanics), and molecular computing, demonstrate that the limits of computation, and therefore of computer models, are still unclear. Furthermore, as technology advances, the perceived limits of computation recede even further. Given this situation, there seems to be no reason to prematurely jump to the conclusion that computer models cannot demonstrate the full behavioral repertoire of any given biological function.

Drug Discovery and Invention

With the preceding in mind, the creation or discovery of new therapeutics to intervene in the evolution and function of biological systems might better be approached by strengthening the analogy between computer models and biological systems. Perhaps it is unwarranted, as yet, to build computers and computer models in part out of biological materials; but, it is not unwarranted to begin expanding modeling methods to cover and account for more of those methods that have evolved and are used by nature to obtain the referent systems, themselves. FURM, by positing methods for the creation, evolution, measurement, and selection of computational representations of functional units in biology is doing just that.

ACKNOWLEDGEMENTS

This work was supported by funding provided by CAH. We thank Michael Roberts not only for essential IPRL data, but also for open and useful discussions. We thank Amina Ann Qutub for accepting the risk of being the first graduate student to see the merit in application of agent-based models to biological systems and to initiate thesis research to explore feasibility (we thank the Whitaker Foundation for providing Ms. Qutub with the Graduate Bioengineering Fellowship that enabled that course of events). We also thank the members of the Biosystems Group—the Hunt lab—for support in these efforts and helpful discussions.

BIBLIOGRAPHY

Akaike, H., Information Theory and an Extension of the Maximum Likelihood Principle, *Proceedings of the Second International Symposium on Information Theory*. 267-281, 1973.

Anissimov, Yuri G., Bracken, Anthony J. and Roberts, Michael S., Interconnected Tubes Model of Hepatic Elimination. J. *Theor.Biol.* **188**:89-101, 1997.

Anissimov, Yuri G., and Roberts, Michael S., A Compartmental Model of Hepatic Disposition Kinetics: 1. Model Development and Application to Linear Kinetics, *J Pharmacokin. Pharmacodynam.* **29**(2): 131-156, 2002.

Barwise, J., and Moss L. *Vicious Circles.* CSLI Publications. 1996.

Bock, Gregoru, and Goode, Jamie A. (Eds.). *'In Silico' Simulation of Biological Processes* (Novartis Foundation Symposium No. 247), Wiley, Chichester, 2002.

Burns, A., Davies, G. *Concurrent Programming.* Addison-Wesley, pp. 1-2, 1993.

De Maesschalck R, Jouan-Rimbaud D, Massart DL. The Mahalanobis distance. *Chemometr. Intell. Lab.* **50**:1-18, 2000.

Duckworth, J. What is chemometrics? In *Proceedindgs of International Conference on Near Infrared Spectroscopy*, 1995. See also <http://www.itl.nist.gov/div898/handbook/pmc/section5/pmc543.htm>

Czarnecki, Krzysztof, and Eisenecker, Ulrich. *Generative Programming: Methods, Tools, and Applications.* Addison-Wesley, pp. 10, 251-254, 2000.

Gumucio, Jorge J. and Miller, Deborah L. Zonal Hepatic Function: Solute-Hepatocyte Interactions Within the Liver Acinus. *Prog. Liver Diseases* **7**:17-30, 1982.

Hung, Daniel Y., Chang, Ping, Weiss, Michael, and Roberts, Michael S. Structure-Hepatic Disposition Relationships for Cationic Drugs in Isolated Perfused Rat Livers: Transmembrane Exchange and Cytoplasmic Binding Process. *J. Pharmacol. Exper. Therap.* **297**(2):780–89, 2001.

Hunter, P.J., Nielsen, P.M.F., and Bullivant, D. The IUPS Physiome Project, in Bock, Gregoru, and Goode, Jamie A. (Eds.). *'In Silico' Simulation of Biological Processes* (Novartis Foundation Symposium No. 247), Wiley, Chichester, 207- 221, 2002.

Jennings, Nicholas R. An Agent-Based Approach For Building Complex Software Systems. *Communications of the ACM* **44**(4):35-41, 2001.

Koo, A, Liang, I.Y., and Cheng, K.K. The terminal hepatic microcirculation in the rat. *Quart. J. Exp. Physiol. Cogn. Med.* **60**:261-266, 1975.

Kullback, S. and Leibler, R. A. On Information and Sufficiency. *Ann. Math. Stat.* **22**:79-86, 1951.

Miller, Deborah L., Zanolli, Caroline S., and Gumucio, Jorge J. Quantitative Morphology of the Sinusoids of the Hepatic Acinus. *Gastroenterology* **76**:965-969, 1979.

Noble, Denis. The Rise of Computational Biology, *Nat. Rev. Mol. Cell. Bio.* **3**:460, 2002.

Roberts, Michael S, and Anissimov, Yuri G., Modeling of Hepatic Elimination and Organ Distribution Kinetics with the Extended Convection-Dispersion Model, *J. Pharmacokin. Biopharm.* **27**(4):343-382, 1999.

Sanchez, Susan M., and Lucas, Thomas W. Exploring the World of Agent-Based Simulations: Simple Models, Complex Analyses, in *Proceedings of the 2002 Winter Simulation Conference,* Yücesan, E. et al., eds., pp. 116-126, 2002.

Schneider, J., Lumpe, M., and Nierstrasz O. Agent Coordination via Scripting Languages. In *Coordination of Internet Agents,* Omicini, Zambonelli, Lusch and Tolksdorf, eds., Springer-Verlag, pp 153-175, 2001.

Sejnowski, Terry, Ed. *Complexity in Biological Information Processing* (Novartis Foundation Symposium 239), Wiley, Chichester, pp. 150-59, 2001.

Smith, B. *On the Origin of Objects*. MIT Press, p.50, 1996,

Stix, Gary. Reverse-Engineering Clinical Biology. *Scientific Ameriacn*, 28-30, February 2003.

Thornton, Stephen; "Karl Popper", The Stanford Encyclopedia of Philosophy (Winter 2002 Edition), Edward N. Zalta (ed.), URL = <http://plato.stanford.edu/archives/win2002/entries/popper/>

Teutsch, Harald F., Schuerfeld, Dirk, and Groezinger, Elke. Three-Dimensional Reconstruction of Parenchymal Units in the Liver of the Rat. *Hepatology* **29**(2): 494-505, 1999.

APPENDIX

Figure A1 illustrates that outflow profiles change as the number of experiments (model runs) is increased from 20, to 60, to 200. Note that an experimental result comprising 20 model runs is analogous to results one might obtain if one could conduct an in vitro perfusion experiment on only 3-to-4 liver lobules. Figure A2 shows results from four independent (i.e., the seed for the random number generator is changed each time) repetitions of 60 experiments each using the parameter settings in Listing 1. Changing the seed alters the specifics for all stochastic parameters (e.g., placement of sinusoids on the digraph), thus providing a unique, individual version of the model. The coefficient of variation (CV) for the six repetitions at each time was calculated. The values were higher at earlier and later times. Between 2 and 80 seconds CV values ranged from 1.2 to 54.4%, averaging 16.1%.

Figure A1. Output of *ArtModel* using the parameter settings in Listing 1. The number of experiments is 20 (open circles), 60 (open squares), and 200 (filled circles). For clarity and to avoid overlap, each open circle value is shifted down by a constant, logC = –0.25, and each open square value is shifted up by logC = 0.25.

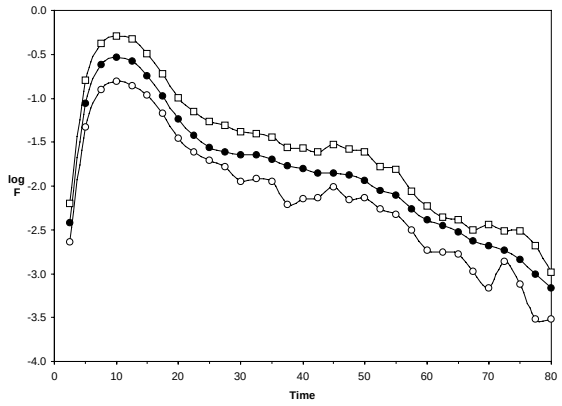


Figure A2. Output of *ArtModel* using the parameter settings in Listing 1. The values shown are for four independent repetitions of 60 experiments each. For clarity and to avoid overlap, the values of three data sets have been shifted by a constant, logC: logC = –0.2 (open squares), +0.2 (open circles), and +0.4 (open diamonds).